RUBY/LDAP TUTORIAL

Ruby/LDAP is an extension library for Ruby. It provides the interface to some LDAP libraries like OpenLDAP, UMich LDAP, Netscape SDK, ActiveDirectory.

The common API for application development is described in RFC1823 and is supported by Ruby/LDAP.

Ruby/LDAP Installation:

You can download and install a complete Ruby/LDAP package from SOURCEFORGE.NET.

Before installing Ruby/LDAP, make sure you have following components:

- Ruby 1.8.x (at least 1.8.2 if you want to use ldap/control).
- OpenLDAP, Netscape SDK, Windows 2003 or Windows XP.

Now you can use standard Ruby Installation method. Before starting, if you'd like to see the available options for extconf.rb, run it with '--help' option.

```
$ ruby extconf.rb [--with-openldap1|--with-openldap2| \
                   --with-netscape|--with-wldap32]
$ make
$ make install
```

NOTE: If you're building the software on Windows, you may need to use nmake instead of make.

Establish LDAP Connection:

This is a two step process:

Step 1: Create Connection Object

Following is the syntax to create a connection to a LDAP directory.

```
LDAP::Conn.new(host='localhost', port=LDAP_PORT)
```

- **host:** This is the host ID running LDAP directory. We will take it as *localhost*
- **port:** This is the port being used for LDAP service. Standard LDAP ports are 636 and 389. Make sure which port is being used at your server otherwise you can use LDAP::LDAP_PORT.

This call returns a new *LDAP::Conn* connection to the server, *host*, on port *port*.

Step 2: Binding

This is where we usually specify the username and password we will use for the rest of the session.

Following is the syntax to bind an LDAP connection, using the DN, **dn**, the credential, **pwd**, and the bind method, **method**

```
conn.bind(dn=nil, password=nil, method=LDAP::LDAP_AUTH_SIMPLE)do
....
end
```

You can use same method without a code block. In this case you would need to unbind the connection explicitly as follows:

```
conn.bind(dn=nil, password=nil, method=LDAP::LDAP_AUTH_SIMPLE)
....
conn.unbind
```

If a code block is given, *self* is yielded to the block.

We can now perform search, add, modify or delete operations inside the block of the bind method (between bind and unbind), provided we have the proper permissions.

Example:

Assuming we are working on a local server, let's put it things together with appropriate host, domain, user id and password etc.

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

conn = LDAP::Conn.new($HOST, $PORT)
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')
....
conn.unbind
```

Adding an LDAP Entry:

Adding an LDPA entry is a two step process:

Step 1: Creating *LDAP::Mod* object

We need *LDAP::Mod* object pass to *conn.add* method to create an entry. Here is a simple syntax to create *LDAP::Mod* object:

```
Mod.new(mod_type, attr, vals)
```

- **mod_type:** One or more option LDAP_MOD_ADD, LDAP_MOD_REPLACE or LDAP_MOD_DELETE.
- **attr:** should be the name of the attribute on which to operate.
- **vals:** is an array of values pertaining to *attr*. If *vals* contains binary data, *mod_type* should be logically OR'ed (|) with LDAP_MOD_BVALUES.

This call return *LDAP::Mod* object which can be passed to methods in the LDAP::Conn class, such as Conn#add, Conn#add_ext, Conn#modify and Conn#modify_ext.

Step 2: Calling *conn.add* Method

Once we are ready with *LDAP::Mod* object, we can call *conn.add* method to create an entry. Here is a syntax to call this method:

```
conn.add(dn, attrs)
```

This method adds an entry with the DN, *dn*, and the attributes, *attrs*. Here *attrs* should be either an array of *LDAP::Mod* objects or a hash of attribute/value array pairs.

Example:

Here is a complete example which will create two directory entries:

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

conn = LDAP::Conn.new($HOST, $PORT)
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')

conn.perror("bind")
entry1 = [
  LDAP.mod(LDAP::LDAP_MOD_ADD,'objectclass',['top','domain']),
  LDAP.mod(LDAP::LDAP_MOD_ADD,'o',['TTSKY.NET']),
  LDAP.mod(LDAP::LDAP_MOD_ADD,'dc',['localhost']),
}

entry2 = [
  LDAP.mod(LDAP::LDAP_MOD_ADD,'objectclass',['top','person']),
  LDAP.mod(LDAP::LDAP_MOD_ADD, 'cn', ['Zara Ali']),
  LDAP.mod(LDAP::LDAP_MOD_ADD | LDAP::LDAP_MOD_BVALUES, 'sn',
                     ['ttate','ALI', "zero\000zero"]),
]

begin
  conn.add("dc=localhost, dc=localdomain", entry1)
  conn.add("cn=Zara Ali, dc=localhost, dc=localdomain", entry2)
rescue LDAP::ResultError
  conn.perror("add")
  exit
end
conn.perror("add")
conn.unbind
```

Modifying an LDAP Entry:

Modifying an entry is similar to adding one. Just call the *modify* method instead of *add* with the attributes to modify. Here is a simple syntax of *modify* method.

```
conn.modify(dn, mods)
```

This method modifies an entry with the DN, *dn*, and the attributes, *mods*. Here *mods* should be either an array of *LDAP::Mod* objects or a hash of attribute/value array pairs.

Example:

To modify the surname of the entry which we added in the previous section, we would write:

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

conn = LDAP::Conn.new($HOST, $PORT)
```

```
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')

conn.perror("bind")
entry1 = [
  LDAP.mod(LDAP::LDAP_MOD_REPLACE, 'sn', ['Mohtashim']),
]

begin
  conn.modify("cn=Zara Ali, dc=localhost, dc=localdomain", entry1)
rescue LDAP::ResultError
  conn.perror("modify")
  exit
end
conn.perror("modify")
conn.unbind
```

Deleting an LDAP Entry:

To delete an entry, call the *delete* method with the distinguished name as parameter. Here is a simple syntax of *delete* method.

```
conn.delete(dn)
```

This method deletes an entry with the DN, *dn*.

Example:

To delete *Zara Mohtashim* entry which we added in the previous section, we would write:

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

conn = LDAP::Conn.new($HOST, $PORT)
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')

conn.perror("bind")
begin
  conn.delete("cn=Zara-Mohtashim, dc=localhost, dc=localdomain")
rescue LDAP::ResultError
  conn.perror("delete")
  exit
end
conn.perror("delete")
conn.unbind
```

Modifying the Distinguished Name:

It's not possible to modify the distinguished name of an entry with the *modify* method. Instead, use the *modrdn* method. Here is simple syntax of *modrdn* method:

```
conn.modrdn(dn, new_rdn, delete_old_rdn)
```

This method modifies the RDN of the entry with DN, *dn*, giving it the new RDN, *new_rdn*. If *delete_old_rdn* is *true*, the old RDN value will be deleted from the entry.

Example:

Suppose we have the following entry:

```
dn: cn=Zara Ali,dc=localhost,dc=localdomain
cn: Zara Ali
sn: Ali
objectclass: person
```

Then we can modify its distinguished name with the following code:

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

conn = LDAP::Conn.new($HOST, $PORT)
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')

conn.perror("bind")
begin
  conn.modrdn("cn=Zara Ali, dc=localhost, dc=localdomain",
              "cn=Zara Mohtashim", true)
rescue LDAP::ResultError
  conn.perror("modrdn")
  exit
end
conn.perror("modrdn")
conn.unbind
```

Performing a Search:

To perform a search on a LDAP directory, use the *search* method with one of three different search modes:

- **LDAP_SCOPE_BASEM:** Search only the base node.
- **LDAP_SCOPE_ONELEVEL:** Search all children of the base node.
- **LDAP_SCOPE_SUBTREE:** Search the whole subtree including the base node.

Example:

Here we are going to search the whole subtree of entry *dc=localhost, dc=localdomain* for *person* objects:

```
#/usr/bin/ruby -w

require 'ldap'

$HOST =    'localhost'
$PORT =    LDAP::LDAP_PORT
$SSLPORT = LDAP::LDAPS_PORT

base = 'dc=localhost,dc=localdomain'
scope = LDAP::LDAP_SCOPE_SUBTREE
filter = '(objectclass=person)'
attrs = ['sn', 'cn']

conn = LDAP::Conn.new($HOST, $PORT)
conn.bind('cn=root, dc=localhost, dc=localdomain','secret')

conn.perror("bind")
begin
  conn.search(base, scope, filter, attrs) { |entry|
     # print distinguished name
     p entry.dn
     # print all attribute names
     p entry.attrs
```

```
    # print values of attribute 'sn'
    p entry.vals('sn')
    # print entry as Hash
    p entry.to_hash
  }
rescue LDAP::ResultError
  conn.perror("search")
  exit
end
conn.perror("search")
conn.unbind
```

This invokes the given code block for each matching entry where the LDAP entry is represented by an instance of the LDAP::Entry class.With the last parameter of search you can specify the attributes in which you are interested, omitting all others. If you pass nil here, all attributes are returned same as "SELECT *" in relational databases.

The dn method(alias for get_dn) of the LDAP::Entry class returns the distinguished name of the entry, and with the to_hash method you can get a hash representation of its attributes (including the distinguished name).To get a list of an entry.s attributes use the attrs method (alias for get_attributes). Also, to get the list of one specific attribute.s values, use the vals method (alias for get_values)

Handling Errors:

Ruby/LDAP defines two different exception classes:

- In case of an error, the new, bind or unbind methods raise an LDAP::Error exception.
- In case of add, modify, delete or searching an LDAP directory raise an LDAP::ResultError.

Further Reading:

For a complete detail on LDAP methods please refer to standard documentation for LDAP Documentation.